Software Engineering Research for Exploratory Research Software

A. Dubey

June 12, 2019

It is well known and understood that software development with some thought given to design and engineering delivers better results than without. It is also well known that a great deal of research has gone into understanding processes that improve software development, and methodologies have been developed based on the outcomes of this research that have been of tremendous benefit. It is further known that the history of the scientific software development is littered with cases where no thought was given to either. A confluence of these three well known facts has given rise to a partially false belief in many quarters that if only we could bring these methodologies and practices to scientific software development, it would solve most problems related to its quality. The fundamental reason that this belief is false is because almost all of software engineering research and methodology development has focused on commercial software where the features and capabilities of the end product are largely known. The research and development departments of even large companies such as Google, BMW, and Cisco have difficulty in fitting the development and lifecycle models that work excellently in their production departments. Any quality shortcomings in their research software are protected from manifesting in their products precisely because no research prototype will turn into a product without going through the proper production pipeline. In scientific software the research prototypes are often used directly as a product, therefore there is no filtering of quality shortcomings. Therefore, a software engineering system and development methodology that ensures quality of software all through research and production stages is a unique challenge for our community, and therefore requires solutions that are tailored for our specific needs.

In the development of scientific software, the expertise map of its various components can be hugely diverse, as also the priorities. Our community has discussed the conflicting priorities of scientific publications versus software quality in various forums for a while. We have yet to come to grips with another aspect of priorities; what is used in science production by the domain scientist can be a research code for a computational mathematician. As an example, the shock hydrodynamics and magnetohydrodynamics solvers are the main workhorses of the FLASH code. Almost all science campaigns use one or the other. However, for the computational mathematician who developed those algorithms, it is the code where he exercises his new ideas. So it is in his interest to build in run-time flexibility in the code so that he can do his own kind of parameter exploration for convergence, stability, constraints on CFL number etc. This runtime flexibility is at odds with maintainable and performant code because often the alternatives are exercised very deep in the call stack. The model of prototyping first and then bringing it into the production code is applicable when the algorithm is new, or the code being developed can be exercised meaningfully in isolation. However, in the fifteen year long history of this code unit, there has never been a time when tweaks were not being done, and many of them needed to be exercised by production level science applications to be characterized and adopted appropriately.

The example above is only one instance of conflicts in technical priorities which are every bit as challenging to solve as the sociological ones. It is also a challenge unique to interdisciplinary research software that computational science requires, and is a research opportunity for the software engineering community. Several other challenges that present research opportunities in software engineering are enumerated below.

1. Repository workflow management for software that has most of its components under continuous research and development.
2. Verification methodology for software meant for exploration where the correct solution is not know apriori.
3. Software lifecycle where there are major capability changes almost continuously over time.
4. Software design for interdisciplinary research software which will not respect the encapsulation boundaries because of domain demands.
5. Quality control and provenance in a constantly changing environment.

These and other questions can become basis for more meaningful engagement between the software engineering research community at large and the computational science software developers. It is my belief that such engagements will also yield insight about management of research software at the enterprise level.